AYI NEDJIMI Consultants

Ghidra : Guide de Reverse Engineering pour Débutants

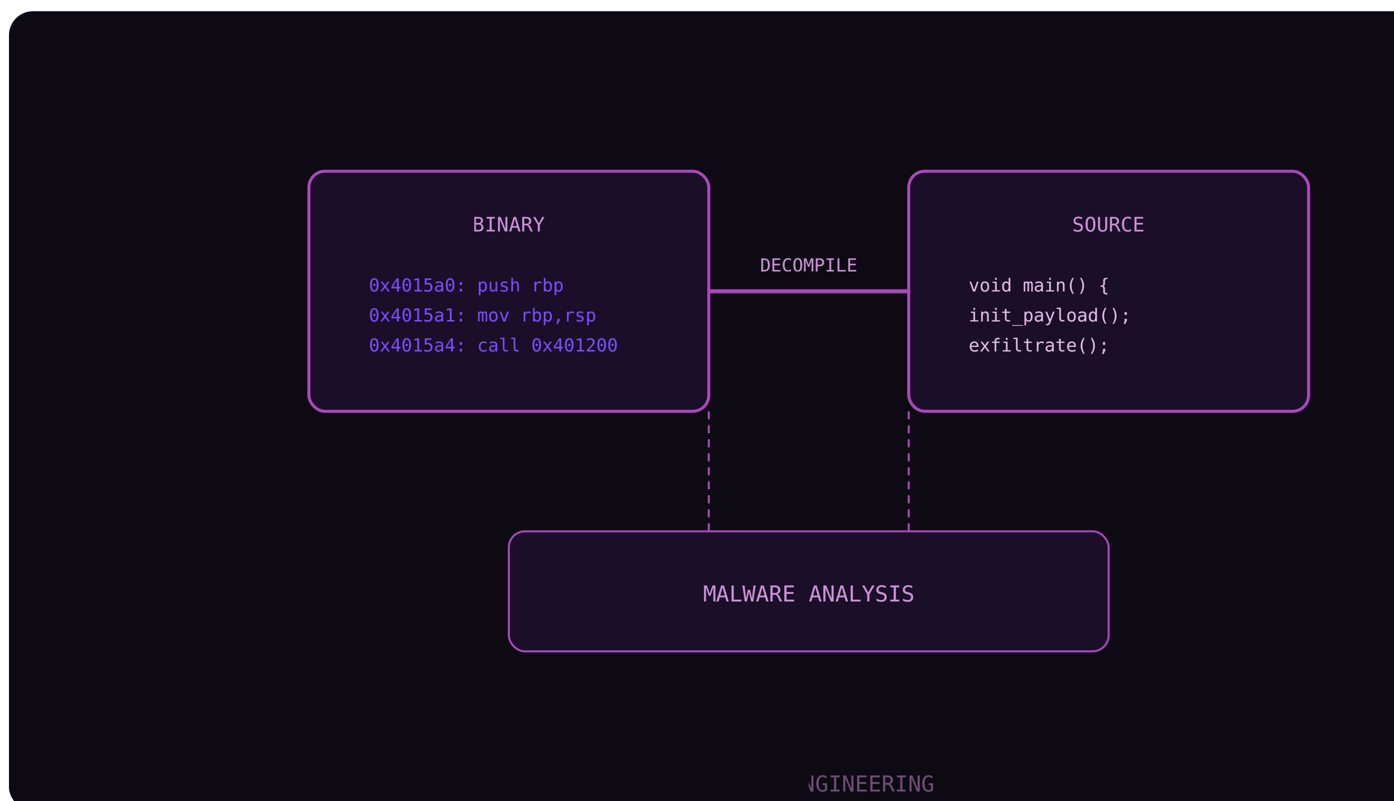
Catégorie : Retro-Ingenierie Lecture : 17 min Publié le : 08/03/2026 Auteur : Ayi NEDJIMI

Guide complet Ghidra pour débutants : installation, interface CodeBrowser, décompilateur, analyse de binaires PE/ELF, scripting Java/Python, exercice.

Ghidra : Guide de Reverse Engineering pour Débutants constitue un enjeu majeur pour les professionnels de la sécurité informatique et les équipes techniques. Guide complet Ghidra pour débutants : installation, interface CodeBrowser, décompilateur, analyse de binaires PE/ELF, scripting Java/Python, exercice. Ce guide détaillé sur ghidra reverse engineering guide debutant propose une méthodologie structurée, des outils éprouvés et des recommandations opérationnelles directement applicables. L'objectif est de fournir aux praticiens — consultants, ingénieurs sécurité, administrateurs systèmes — les connaissances et les techniques nécessaires pour aborder ce sujet avec rigueur. Chaque section s'appuie sur des retours d'expérience terrain et intègre les évolutions les plus récentes du domaine. Les recommandations présentées sont adaptées aux environnements d'entreprise et tiennent compte des contraintes opérationnelles réelles.

Avertissement : Les techniques présentées dans cet article sont destinées exclusivement à des fins éducatives et de tests autorisés. Toute utilisation malveillante est illégale et contraire à l'éthique professionnelle.

1. Introduction : pourquoi apprendre le reverse engineering avec Ghidra



Le reverse engineering -- ou rétro-ingénierie -- constitue l'une des compétences les plus fondamentales en cybersécurité. Qu'il s'agisse d'analyser un **malware inconnu**, de comprendre le fonctionnement interne d'un protocole propriétaire, ou de résoudre un challenge CTF, la capacité à lire et interpréter du code assembleur décompilé est un atout décisif. Depuis sa publication par la **NSA en mars 2019** sous licence Apache 2.0, Ghidra a change l'accès aux outils de reverse engineering professionnels en offrant gratuitement des fonctionnalités qui rivalisent avec IDA Pro, dont la licence coûte plusieurs milliers d'euros. Ce guide approfondi examine en detail les aspects fondamentaux et avances de Ghidra, en proposant une analyse structuree et documentee des enjeux actuels. Les professionnels y trouveront des recommandations concretes, des methodologies eprouvees et des retours d'experience terrain directement applicables en environnement de production.

Points cles :

- 1. Introduction : pourquoi apprendre le reverse engineering avec Ghidra
- 2. Présentation de Ghidra : origines et architecture
- 3. Installation et configuration multi-plateforme
- 4. L'interface de Ghidra : maîtriser le CodeBrowser
- 5. Premier projet : importer et analyser un binaire

Ghidra n'est pas un simple désassembleur : c'est un **framework d'analyse de binaires complet**, doté d'un décompilateur multi-architecture, d'un moteur de scripting puissant, d'un système de collaboration, et d'une architecture extensible par plugins. Son décompilateur, basé sur le langage intermédiaire **P-Code**, produit un pseudo-code C remarquablement lisible qui transforme radicalement la productivité de l'analyste. Des centaines d'architectures processeur sont supportées : x86, x86-64, ARM, MIPS, PowerPC, RISC-V, AVR, et bien d'autres. Pour plus d'informations, consultez les ressources de ANSSI.

Ce guide vous accompagne pas à pas, de l'installation à l'analyse de votre premier binaire. Nous couvrirons l'interface CodeBrowser, les techniques de navigation dans le code, le décompilateur, le scripting, et un exercice pratique complet sur un crackme CTF. Si vous travaillez déjà en analyse de malwares ou en forensique mémoire, Ghidra deviendra rapidement un outil indispensable de votre arsenal. Pour plus d'informations, consultez les ressources de MITRE ATT&CK.

Pourquoi Ghidra plutôt qu'un autre outil ? Sa gratuité, sa communauté active, son décompilateur intégré et son support multi-architecture en font le choix idéal pour débuter. Même les analystes chevronnés qui utilisent IDA Pro intègrent Ghidra dans leur workflow pour ses capacités de scripting et sa flexibilité.

Usage responsable

Le reverse engineering est un outil puissant qui doit être utilisé dans un cadre légal. En France, l'article L122-6-1 du Code de la propriété intellectuelle autorise la décompilation à des fins d'interopérabilité et de sécurité. Utilisez toujours Ghidra dans un contexte autorisé : analyse de malwares, recherche de vulnérabilités avec accord, CTF, ou étude de logiciels libres.

Notre avis d'expert

L'analyse de malware est un art qui requiert patience et méthodologie. Chaque échantillon raconte une histoire — les techniques d'obfuscation utilisées, les C2 contactés, les mécanismes de persistance déployés. Décoder cette histoire est essentiel pour construire des défenses efficaces.

Disposez-vous en interne des compétences de rétro-ingénierie nécessaires pour analyser un malware ciblant votre organisation ?

2. Présentation de Ghidra : origines et architecture

2.1 Genèse d'un outil de la NSA

Ghidra a été développé en interne par la **National Security Agency (NSA)** pendant plus de vingt ans avant sa publication open source lors de la conférence RSA 2019. L'outil était utilisé quotidiennement par les analystes de la NSA et de ses partenaires dans la communauté du renseignement américain (IC -- Intelligence Community). Sa publication a été motivée par la volonté de démocratiser l'accès à des outils d'analyse avancés et de bénéficier des contributions de la communauté open source.

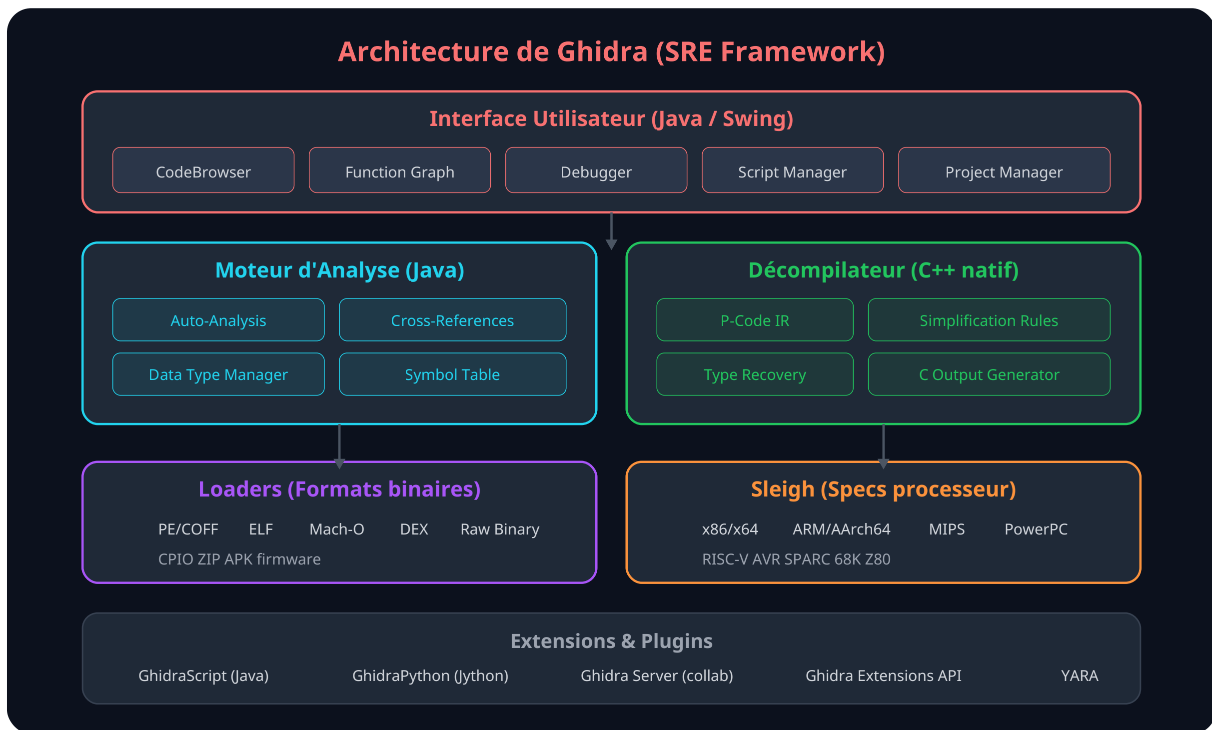
Le projet est hébergé sur GitHub sous l'organisation NationalSecurityAgency et bénéficie d'une communauté très active. Depuis sa sortie, Ghidra a reçu plus de **52 000 étoiles GitHub**, faisant de lui l'un des projets de cybersécurité les plus populaires de la plateforme. Les mises à jour sont régulières, avec des améliorations significatives du décompilateur, le support de nouvelles architectures, et l'ajout de fonctionnalités comme le debugger intégré (depuis Ghidra 10.0) et le support des fichiers PDB amélioré.

2.2 Architecture logicielle

Ghidra est écrit principalement en **Java**, ce qui lui confère une portabilité native sur Windows, Linux et macOS. Le décompilateur, cependant, est un composant C++ compilé nativement (`decompile`) qui communique avec l'interface Java via un protocole XML. Cette architecture hybride offre le meilleur des deux mondes : la portabilité de Java pour l'interface et les performances du C++ pour le moteur de décompilation.

Les composants principaux sont :

- **Ghidra Project Manager** : gestionnaire de projets, point d'entrée de l'application
- **CodeBrowser** : l'outil d'analyse principal avec listing, décompilateur, graphes
- **Sleigh** : le langage de description d'architectures processeur (spécifications d'instructions)
- **P-Code** : le langage intermédiaire utilisé par le décompilateur (Intermediate Representation)
- **Ghidra Server** : serveur de collaboration multi-utilisateurs pour les projets partagés
- **Debugger** : debugger intégré supportant GDB, WinDbg et LLDB (depuis 10.x)
- **Script Manager** : moteur d'exécution pour les scripts Java et Python (Jython/GhidraPython)



3. Installation et configuration multi-plateforme

3.1 Prérequis système

Ghidra nécessite un **Java Development Kit (JDK) 17 ou supérieur**. Depuis Ghidra 11.x, le JDK 21 est recommandé. Les distributions AdoptOpenJDK (Eclipse Temurin), Amazon Corretto ou Oracle JDK sont toutes compatibles. Voici les prérequis minimaux :

Composant	Minimum	Recommandé
JDK	17	21 (LTS)
RAM	4 Go	16 Go+
Disque	1 Go (Ghidra) + espace projets	SSD recommandé
OS	Windows 10+, Linux (64-bit), macOS 10.13+	Linux 64-bit
Résolution	1280x1024	1920x1080+

3.2 Installation pas à pas

Linux (recommandé pour l'analyse de malwares)

```
# 1. Installer JDK 21
sudo apt update && sudo apt install openjdk-21-jdk -y
java -version  # Vérifier l'installation

# 2. Télécharger Ghidra depuis GitHub
wget https://github.com/NationalSecurityAgency/ghidra/releases/download/Ghidra_11.3_build/
ghidra_11.3_PUBLIC_20250115.zip

# 3. Extraire l'archive
unzip ghidra_11.3_PUBLIC_20250115.zip -d /opt/
ln -s /opt/ghidra_11.3_PUBLIC /opt/ghidra

# 4. Lancer Ghidra
/opt/ghidra/ghidraRun

# 5. (Optionnel) Créer un alias
echo 'alias ghidra="/opt/ghidra/ghidraRun"' >> ~/.bashrc
source ~/.bashrc
```

Windows

```
# PowerShell - Installation JDK via winget
winget install EclipseAdoptium.Temurin.21.JDK

# Configurer JAVA_HOME (si nécessaire)
[Environment]::SetEnvironmentVariable("JAVA_HOME", "C:\Program Files\Eclipse
Adoptium\jdk-21", "Machine")

# Extraire ghidra_11.3_PUBLIC.zip dans C:\Tools\
# Lancer C:\Tools\ghidra_11.3_PUBLIC\ghidraRun.bat
```

macOS

```
# Installer via Homebrew
brew install openjdk@21
brew install --cask ghidra

# Ou manuellement
export JAVA_HOME=$(/usr/libexec/java_home -v 21)
# Télécharger et extraire depuis GitHub
# Lancer : ./ghidraRun
```

Environnement isolé pour l'analyse de malwares

Si vous analysez des échantillons malveillants, utilisez toujours une **machine virtuelle isolée** (VirtualBox, VMware, QEMU). Désactivez le réseau, utilisez des snapshots, et ne partagez pas de dossiers avec l'hôte. Consultez notre article sur les rootkits kernel-mode pour comprendre pourquoi l'isolation est critique.

Cas concret
L'analyse du malware Pegasus par le Citizen Lab et Amnesty International a révélé un arsenal d'exploitation zero-click ciblant iOS. La rétro-ingénierie des exploits FORCEDENTRY a montré une utilisation innovante de fichiers PDF malveillants traités par le moteur de rendu d'iMessage, sans aucune interaction de la victime.

4. L'interface de Ghidra : maîtriser le CodeBrowser

4.1 Le Project Manager

Au lancement, Ghidra affiche le **Project Manager**, le hub central de gestion de vos projets d'analyse. Un projet Ghidra est un conteneur qui regroupe un ou plusieurs binaires importés, leurs analyses, vos annotations et vos scripts. Deux types de projets existent :

- **Non-Shared Project** : projet local, stocké dans un répertoire sur votre disque. Idéal pour le travail individuel.
- **Shared Project** : projet hébergé sur un Ghidra Server, permettant la collaboration multi-analystes avec un système de verrouillage et de versioning.

Pour créer un projet, cliquez sur `File > New Project`, choisissez le type, nommez-le et sélectionnez un répertoire. Vous pouvez ensuite importer des binaires via `File > Import File` ou par glisser-déposer. Ghidra détecte automatiquement le format (PE, ELF, Mach-O, DEX, etc.) et l'architecture processeur.

4.2 Le CodeBrowser : votre plan de travail

Le **CodeBrowser** est l'outil d'analyse principal. Il s'ouvre en double-cliquant sur un fichier importé et analysé. Son interface est organisée en panneaux que vous pouvez réorganiser librement. Les panneaux essentiels sont :

Listing (panneau central)
Le listing affiche le désassemblage du binaire. Chaque ligne représente une instruction assembleur avec son adresse, ses octets bruts, le mnémonique et les opérandes. Les commentaires, labels et cross-references sont affichés en surbrillance. Les raccourcis essentiels :

- `G` : aller à une adresse spécifique
- `L` : renommer un label ou une fonction
- `;` : ajouter un commentaire EOL (end-of-line)
- `Ctrl+Shift+;` : ajouter un commentaire plate
- `/` : ajouter un commentaire repeated
- `T` : changer le type d'une donnée
- `D` : définir une donnée (byte, word, dword, string...)
- `C` : convertir en code (désassembler)
- `F` : créer une fonction à l'adresse courante

Decompiler (panneau droit)
Le **décompilateur** affiche en temps réel le pseudo-code C de la fonction sélectionnée dans le listing. C'est l'un des atouts majeurs de Ghidra : il traduit automatiquement l'assembleur en code C lisible. Chaque variable, paramètre et appel de fonction est interactif -- vous pouvez renommer, retyper et commenter directement dans le panneau de décompilation, et les modifications se propagent dans le listing.

Function Graph
Le graphe de flux de contrôle (CFG) affiche visuellement les blocs de base d'une fonction et leurs connexions. Les branches conditionnelles sont colorées (vert pour le saut pris, rouge pour le non-pris). Cet affichage est essentiel pour comprendre la logique d'un algorithme, identifier les boucles et les structures conditionnelles complexes.

Autres panneaux utiles

- **Symbol Tree** : arborescence des imports, exports, fonctions, labels, classes et namespaces
- **Data Type Manager** : bibliothèque de structures, enums et typedefs
- **Bytes** : vue hexadécimale des octets bruts du fichier
- **Console** : sortie des scripts et messages du système
- **Bookmarks** : marque-pages pour naviguer rapidement entre les points d'intérêt

Savez-vous identifier les techniques d'anti-analyse utilisées par les malwares modernes ?

5. Premier projet : importer et analyser un binaire

5.1 Importation d'un fichier PE (Windows)

Pour importer un exécutable Windows (`.exe` ou `.dll`), utilisez `File > Import File`. Ghidra lance automatiquement ses analyseurs de format et affiche une boîte de dialogue avec les informations détectées :

- **Format** : Portable Executable (PE)
- **Language** : x86:LE:64:default (pour un PE 64-bit) ou x86:LE:32:default (32-bit)
- **Compiler** : VisualStudio:default, gcc, etc.
- **Destination Folder** : emplacement dans le projet

Cliquez sur `Options...` pour configurer les options d'import avancées. Les options les plus utiles incluent :

- **Load External Libraries** : tente de résoudre les imports dynamiques (DLL)
- **Apply Processor Defined Labels** : applique les noms symboliques connus du processeur
- **Create Bookmarks** : crée des marque-pages pour les points d'intérêt

5.2 Importation d'un fichier ELF (Linux)

Le processus est identique pour les binaires Linux au format ELF. Ghidra détecte automatiquement l'architecture (x86-64, ARM, MIPS, etc.) et le format. Un avantage des binaires ELF non-strippés : les **symboles de debug** (DWARF) sont automatiquement analysés, fournissant les noms de fonctions, les types de variables et les numéros de ligne du code source original.

```
# Pour préparer un binaire de test
gcc -o hello hello.c             # Binaire avec symboles
strip hello -o hello_stripped     # Binaire sans symboles (plus réaliste)
gcc -g -o hello_debug hello.c    # Binaire avec symboles DWARF complets
```

5.3 Auto-Analysis : l'analyse automatique

Après l'import, Ghidra propose de lancer l'**auto-analysis**. Cette étape est cruciale : elle exécute une série d'analyseurs qui identifient les fonctions, résolvent les appels, propagent les types et détectent les structures. Les analyseurs principaux sont :

Analyseur	Fonction	Impact
Disassemble Entry Points	Désassemble à partir des points d'entrée connus	Essentiel
Subroutine References	Identifie les appels de sous-routines	Essentiel
Stack Analysis	Analyse les frames de pile et les variables locales	Important
Function Start Search	Recherche heuristique de prologues de fonctions	Important pour les binaires strippés
Decompiler Parameter ID	Identifie les paramètres des fonctions via le décompilateur	Améliore significativement la décompilation
Windows x86 PE RTTI Analyzer	Analyse les informations RTTI C++	Utile pour les binaires C++
Aggressive Instruction Finder	Recherche agressivement du code non référencé	Peut créer des faux positifs

Conseil : ne désactivez pas l'auto-analysis
Sauf cas particulier (binaire obfusqué, firmware exotique), laissez tous les analyseurs activés par défaut. L'analyse complète peut prendre de quelques secondes à plusieurs minutes selon la taille du binaire. Les résultats sont stockés dans la base de données du projet et n'ont pas besoin d'être recalculés. Pour l'analyse de binaires obfusqués, consultez notre guide sur la déobfuscation de malwares polymorphes.

6. Navigation avancée : symboles, cross-references et recherche

6.1 Symboles et Symbol Tree

Le panneau **Symbol Tree** est votre boussole dans le code. Il organise tous les éléments nommés du binaire en catégories :

- **Imports** : fonctions importées depuis des bibliothèques externes (kernel32.dll, libc.so...)
- **Exports** : fonctions exportées par le binaire (pour les DLL/SO)
- **Functions** : toutes les fonctions identifiées, nommées ou non (FUN_00401000, etc.)
- **Labels** : labels créés manuellement ou automatiquement

- **Classes** : classes C++ détectées via RTTI ou analyse vtable
- **Namespaces** : espaces de noms (C++, .NET)

Un clic sur n'importe quel symbole vous transporte immédiatement à son adresse dans le listing. Le filtrage en temps réel (`Filter`) vous permet de rechercher rapidement dans des binaires contenant des milliers de symboles.

6.2 Cross-References (XRefs) : suivre le flux de données

Les **cross-references** sont l'outil le plus puissant pour comprendre comment le code est connecté. Pour chaque adresse, Ghidra maintient la liste de toutes les références *vers* et *depuis* cette adresse. Types de références :

- **CALL** : appel de fonction (`call 0x401000`)
- **UNCONDITIONAL_JUMP** : saut inconditionnel (`jmp`)
- **CONDITIONAL_JUMP** : saut conditionnel (`je` , `jne` , etc.)
- **DATA** : référence à une donnée (lecture/écriture de variable globale)
- **READ** / **WRITE** : accès en lecture ou écriture à une adresse mémoire

Pour afficher les XRefs d'un symbole, placez le curseur dessus et appuyez sur `Ctrl+Shift+F` (Find References to). La fenêtre résultante liste toutes les locations du binaire qui référencent ce symbole. C'est ainsi que vous pouvez tracer le parcours d'une chaîne de caractères suspecte, d'une clé de chiffrement, ou d'un appel API critique comme `CreateRemoteThread` utilisé dans les techniques d'escalade de privilèges Windows.

6.3 Recherche dans le binaire

Ghidra offre plusieurs méthodes de recherche :

- **Search > Memory** : recherche de motifs d'octets, de chaînes ou d'expressions régulières dans la mémoire du programme
- **Search > For Strings** : extrait toutes les chaînes ASCII, UTF-8 et Unicode détectées
- **Search > For Scalars** : recherche de valeurs numériques constantes (utile pour trouver des magic numbers, des tailles de buffer, etc.)
- **Search > For Instruction Patterns** : recherche de séquences d'instructions spécifiques
- **Search > For Address Tables** : détecte les tables de pointeurs (vtables, jump tables)

La recherche de chaînes est souvent le premier réflexe d'un analyste. Les chaînes révèlent des messages d'erreur, des URL de C2, des clés de registre, des chemins de fichiers, et des indicateurs de compromission. Pour une approche systématique de la recherche d'IOC dans les binaires, notre article sur les frameworks d'analyse de malwares par IA détaille des techniques complémentaires.

7. Le décompilateur : du binaire au pseudo-code C

7.1 Le langage intermédiaire P-Code

Le décompilateur de Ghidra repose sur le concept de **P-Code** (Processor Code), un langage intermédiaire (IR -- Intermediate Representation) qui abstrait les spécificités de chaque architecture processeur. Chaque instruction assembleur native est traduite en une séquence d'opérations P-Code élémentaires. Ce mécanisme est la clé de la portabilité du décompilateur : ajouter le support d'une nouvelle architecture revient à écrire la traduction en P-Code via le langage **Sleigh**, sans modifier le moteur de décompilation lui-même.

Le pipeline de décompilation suit ces étapes :

1. **Traduction en P-Code** : les instructions natives sont converties en opérations P-Code
2. **Construction du SSA** : transformation en forme SSA (Static Single Assignment) pour l'analyse de flux de données
3. **Simplification** : application de règles de simplification algébrique et logique
4. **Propagation de types** : inférence des types de variables à partir des opérations et des signatures connues
5. **Restructuration du flux de contrôle** : reconstruction des if/else, switch/case, boucles while/for
6. **Génération du code C** : production du pseudo-code C final lisible

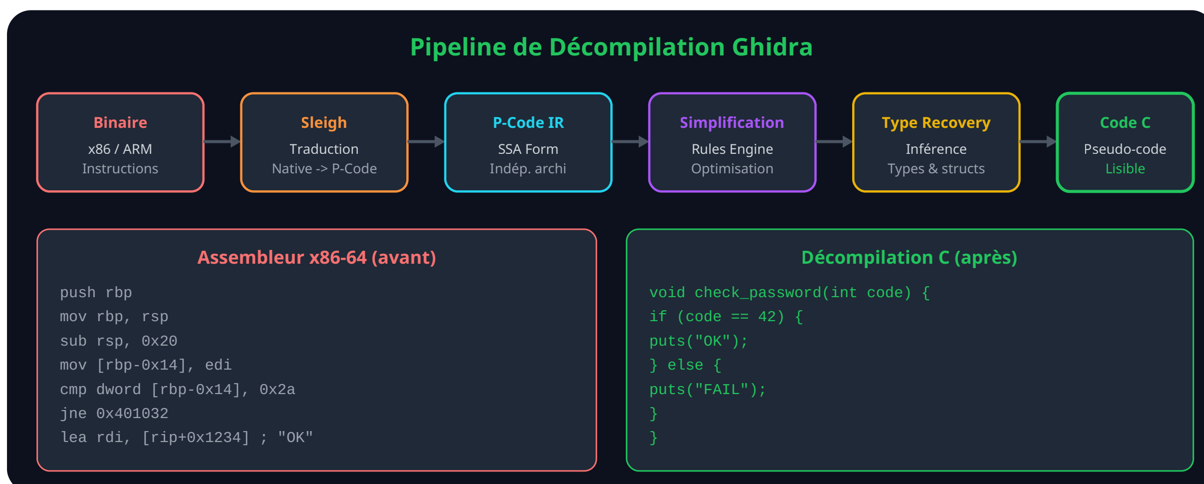
7.2 Améliorer la décompilation

La qualité de la décompilation dépend fortement des informations que vous fournissez à Ghidra. Voici les actions qui améliorent le plus le résultat :

Retyper les variables et paramètres
Le décompilateur nomme les variables `local_10`, `param_1`, etc. En les renommant et en leur attribuant le bon type, le code devient beaucoup plus lisible. Cliquez droit sur une variable > `Retype Variable` (ou `Ctrl+L`) et choisissez le type approprié (char*, DWORD, HANDLE, struct...). La propagation est automatique.

Définir les signatures de fonctions
Modifier le prototype d'une fonction (`Edit Function Signature`) en spécifiant ses paramètres et son type de retour améliore en cascade toutes les fonctions qui l'appellent. Utilisez les données types de Ghidra (`Windows/DataTypes`) pour les API Windows, ou importez des fichiers d'en-tête C.

Appliquer des structures
Quand le code accède à des champs via des offsets (`*(param_1 + 0x28)`), définir une structure avec `Data Type Manager` transforme ces accès obscurs en `pContext->ProcessId`, rendant le code immédiatement compréhensible.



8. Annotations, renommage et documentation

8.1 L'art de l'annotation

L'analyse de binaires est un processus itératif. Vous ne comprendrez pas tout au premier passage. Les annotations sont votre mémoire externe : elles documentent vos hypothèses, vos découvertes et vos questions. Ghidra propose plusieurs types de commentaires :

- **EOL Comments** (`;`) : commentaires en fin de ligne, visibles directement dans le listing
- **Pre Comments** : commentaires affichés avant l'instruction, utiles pour documenter un bloc logique
- **Post Comments** : commentaires après l'instruction
- **Plate Comments** (`Ctrl+Shift+;`) : commentaires encadrés, idéaux pour les en-têtes de fonctions
- **Repeatable Comments** (`/`) : commentaires qui se propagent automatiquement via les cross-references

8.2 Stratégie de renommage

Un binaire strippé présente des fonctions nommées `FUN_00401000`, `FUN_00401050`, etc. Renommer ces fonctions au fur et à mesure de votre compréhension transforme un code cryptique en documentation vivante. Conventions recommandées :

- Préfixer par le rôle : `init_`, `parse_`, `encrypt_`, `send_`, `check_`
- Utiliser le snake_case pour les fonctions : `decrypt_c2_payload`
- Préfixer les variables globales par `g_` : `g_config_buffer`
- Annoter les structures par leur rôle : `MALWARE_CONFIG`, `C2_PACKET`
- Marquer les fonctions non analysées avec `TODO_` : `TODO_unknown_init`

Ces conventions s'alignent avec les pratiques décrites dans notre article sur les techniques anti-reverse engineering des APT, qui détaille les techniques d'obfuscation que les malwares emploient pour résister à ce type d'analyse.

9. Scripting : automatiser l'analyse avec Java et Python

9.1 GhidraScript en Java

Ghidra expose une API riche accessible via des scripts Java. Le **Script Manager** (`Window > Script Manager`) liste les scripts disponibles et permet d'en créer de nouveaux. Un script Ghidra hérite de la classe `GhidraScript` et a accès à tout le modèle de données du programme : fonctions, instructions, références, types, etc.

```
// Exemple : lister toutes les fonctions qui appellent CreateRemoteThread
// @category Analysis
// @author Ayi NEDJIMI

import ghidra.app.script.GhidraScript;
import ghidra.program.model.symbol.*;
import ghidra.program.model.listing.*;

public class FindCreateRemoteThread extends GhidraScript {
    @Override
    protected void run() throws Exception {
        SymbolTable symTab = currentProgram.getSymbolTable();
        SymbolIterator symbols = symTab.getSymbolIterator("CreateRemoteThread", true);

        while (symbols.hasNext()) {
            Symbol sym = symbols.next();
            Reference[] refs = getReferencesTo(sym.getAddress());

            for (Reference ref : refs) {
                Function caller = getFunctionContaining(ref.getFromAddress());
                if (caller != null) {
                    println("CreateRemoteThread appelée depuis : " +
                            caller.getName() + " @ " + ref.getFromAddress());
                }
            }
        }
    }
}
```

9.2 GhidraPython (Jython)

Pour les scripts plus rapides à écrire, Ghidra intègre un interpréteur **Jython** (Python 2.7 exécuté dans la JVM). Depuis Ghidra 11.x, le support de Python 3 via **Pyhidra** (cpython bridge) est également disponible. L'API est identique à celle de Java :

```python
# Exemple Python : extraire toutes les chaînes et leur XRefs
# @category Strings
# @author Ayi NEDJIMI

from ghidra.program.model.data import StringDataType
from ghidra.program.util import DefinedDataIterator

data_iterator = DefinedDataIterator.definedStrings(currentProgram)

for data in data_iterator:
    value = data.getValue()
    if value and len(str(value)) > 4:
        refs = getReferencesTo(data.getAddress())
        if refs:
            callers = []
            for ref in refs:
                func = getFunctionContaining(ref.getFromAddress())
                if func:
                    callers.append(func.getName())
            if callers:
                print("String: '{}' @ {} -> Called from: {}".format(
                    value, data.getAddress(), ", ".join(set(callers))))
```

9.3 Scripts essentiels de la communauté

La communauté Ghidra a produit de nombreux scripts utiles :

- **FindCrypt** : détecte les constantes cryptographiques (AES S-Box, SHA tables, RSA)
- **ghidra-scripts (AllSafe)** : collection de scripts pour l'analyse de malwares
- **GhidraOllvm** : simplifie les binaires protégés par OLLVM (Obfuscator-LLVM)
- **ghidra2frida** : génère des hooks Frida à partir de l'analyse Ghidra
- **YARA-GhidraPlugin** : exécute des règles YARA directement depuis Ghidra

10. Plugins essentiels et extensions

10.1 Plugins intégrés

Ghidra est livré avec de nombreux plugins activables via `File > Configure` dans le CodeBrowser. Les plus utiles :

- **Function ID** (FidDb) : identifie les fonctions de bibliothèques standard (libc, MSVCRT, OpenSSL) par leurs patterns d'octets. Similaire au système FLIRT d'IDA Pro.
- **Version Tracking** : compare deux versions d'un même binaire pour identifier les différences (diff binaire). Utile pour l'analyse de patchs de sécurité.
- **BSim** : recherche de similarité entre fonctions à grande échelle, basé sur les caractéristiques du décompilateur.
- **Debugger** : debugger intégré supportant GDB (Linux), WinDbg (Windows) et LLDB (macOS). Permet le debug statique + dynamique unifié.
- **Emulator** : émulateur P-Code intégré pour l'exécution symbolique de fragments de code.

10.2 Extensions tierces recommandées

Extension	Fonction	Cas d'usage
ghidra-delinker-extension	Exporte des fonctions en fichiers objet recompilables	Patching, modding
GhidraBridge	Pont Python 3 (CPython) vers l'API Ghidra	Scripts modernes, ML integration
Kaiju	Analyse de malware par CERT/CC (Carnegie Mellon)	Hashing de fonctions, triage
ret-sync	Synchronisation avec un debugger externe (x64dbg, WinDbg)	Analyse dynamique + statique
GhidraEmu	Émulation avancée de fonctions	Déobfuscation, résolution de strings
Semgrep-Ghidra	Recherche de patterns dans le code décompilé	Détection de vulnérabilités

L'installation d'extensions se fait via `File > Install Extensions` dans le Project Manager, ou manuellement en copiant le dossier de l'extension dans `$GHIDRA_INSTALL_DIR/Ghidra/Extensions`. Pour les développeurs d'extensions, le système **Gradle** de Ghidra facilite la compilation contre la bonne version de l'API.

11. Exercice pratique : résoudre un Crackme CTF

11.1 Contexte du challenge

Appliquons tout ce que nous avons appris sur un crackme simple. Le binaire est un exécutable ELF 64-bit Linux qui demande un mot de passe. Notre objectif : trouver le mot de passe correct sans exécuter le programme. Ce type d'exercice est similaire aux challenges que l'on retrouve dans les compétitions CTF et dans la formation initiale des analystes malware.

11.2 Démarche d'analyse pas à pas

Etape 1 : Import et analyse automatique

```
# Informations initiales avec file et strings
file crackme_easy
# crackme_easy: ELF 64-bit LSB executable, x86-64, dynamically linked

strings crackme_easy | grep -i pass
# Enter password:
# Correct! Well done.
# Wrong password, try again.
```

Importez le fichier dans Ghidra, acceptez l'auto-analysis. Une fois terminée, naviguez vers le **Symbol Tree > Functions**.

Etape 2 : Localiser la fonction main

Dans le Symbol Tree, cherchez `main` ou `entry`. Si le binaire est strippé, naviguez vers l'entry point et suivez les appels jusqu'à trouver la fonction qui appelle `printf` / `puts` avec le message "Enter password".

Etape 3 : Analyser la logique avec le décompilateur

```
// Décompilation typique d'un crackme simple
int main(int argc, char **argv) {
    char user_input[64];
    char *secret = "s3cr3t_k3y_2026";

    puts("Enter password: ");
    fgets(user_input, 64, stdin);

    // Supprimer le newline
    user_input[strcspn(user_input, "\n")] = '\0';

    if (strcmp(user_input, secret) == 0) {
        puts("Correct! Well done.");
        return 0;
    }
    puts("Wrong password, try again.");
    return 1;
}
```

Dans ce cas simple, le mot de passe est visible en clair : `s3cr3t_k3y_2026`. En réalité, les crackmes plus avancés utilisent des transformations (XOR, algorithmes custom, hashing), mais la démarche reste identique :

1. Localiser la fonction de vérification via les chaînes de succès/échec
2. Comprendre la logique de transformation de l'input utilisateur
3. Identifier la valeur attendue ou inverser la transformation
4. Valider en traçant le flux de données complet

11.3 Techniques avancées pour les crackmes complexes

Pour les challenges plus élaborés, combinez Ghidra avec d'autres outils :

- **Debugger Ghidra + breakpoints** : placez un breakpoint sur la comparaison pour inspecter les valeurs en mémoire
- **Emulation P-Code** : émulez la fonction de transformation pour obtenir le résultat attendu
- **Scripting** : automatisez l'extraction et le décryptage des constantes
- **Z3 / Angr** : pour les contraintes complexes, utilisez un solveur SMT en complément

Les compétitions CTF sont le meilleur terrain d'entraînement. Les plateformes comme **crackmes.one**, **root-me.org**, **picoCTF** et **Hack The Box** proposent des centaines de challenges de reverse engineering de difficulté croissante. Pour une approche plus orientée malware réel, consultez notre article sur les malwares mobiles et l'IA.

12. Comparaison : Ghidra vs IDA Pro vs Binary Ninja vs Cutter

Le choix d'un outil de reverse engineering dépend de vos besoins, votre budget et votre workflow. Voici une comparaison détaillée des quatre principaux outils du marché en 2026 :

Critère	Ghidra	IDA Pro	Binary Ninja	Cutter (rizin)
Prix	Gratuit (Apache 2.0)	1 800 - 6 500 EUR/an	349 - 2 499 USD	Gratuit (GPL)
Décompilateur	Intégré (P-Code, excellent)	Hex-Rays (référence, payant séparé)	HLIL (très bon)	r2ghidra (port du décompilateur Ghidra)
Architectures	30+ (le plus large)	20+ (payant par archi)	15+ (en expansion)	20+ (via rizin)
Scripting	Java, Jython, Python 3 (Pyhidra)	IDAPython (Python 3), IDC	Python 3, API riche	Python, JavaScript, r2pipe
Collaboration	Ghidra Server (intégré)	Lumina (cloud), TeamIDA	Enterprise (payant)	Non intégré
Debugger	Intégré (GDB, WinDbg, LLDB)	Intégré (le meilleur)	Intégré (basique)	Intégré via rizin
Performance	Bonne (Java, peut être lent sur gros binaires)	Excellente (C++)	Excellente (C++/Rust)	Bonne (C/Qt)
Communauté	Très active (GitHub, 52k stars)	Historique, plugins matures	Active, croissante	Active (radare2/rizin)
Courbe d'apprentissage	Moyenne	Élevée mais bien documentée	Faible (UI moderne)	Élevée (héritage CLI radare2)

Notre recommandation
Débutants : commencez par Ghidra. Gratuit, complet, bien documenté, avec un décompilateur intégré de qualité professionnelle. **Professionnels** : combinez Ghidra et IDA Pro. IDA reste la référence pour la performance et le debugger, Ghidra complète pour le scripting, la collaboration et l'analyse multi-architecture. **Développeurs** : Binary Ninja offre l'API la plus propre et la meilleure expérience de développement de plugins.

13. Checklist du débutant en reverse engineering

Utilisez cette checklist comme guide de progression. Chaque point maîtrisé vous rapproche du niveau d'un analyste opérationnel :

Checklist du Débutant Ghidra

Phase 1 : Fondamentaux

- [ ] Installer Ghidra + JDK 21
- [ ] Créer un projet, importer un binaire
- [ ] Naviguer dans le CodeBrowser
- [ ] Utiliser le décompilateur
- [ ] Renommer fonctions et variables
- [ ] Suivre les cross-references (XRefs)
- [ ] Résoudre un crackme simple

Phase 2 : Intermédiaire

- [ ] Maîtriser x86/x64 assembleur
- [ ] Définir des structures et types
- [ ] Ecrire des scripts GhidraScript
- [ ] Utiliser Function ID (FidDb)
- [ ] Analyser un binaire C++ (vtables)
- [ ] Identifier des patterns crypto
- [ ] Résoudre un crackme moyen (CTF)

Phase 3 : Avancé -- Analyse de Malwares

- [ ] Analyser un échantillon malveillant réel
- [ ] Déobfusquer du code (XOR, packing)
- [ ] Ecrire des règles YARA depuis Ghidra
- [ ] Utiliser le debugger + émulateur
- [ ] Développer une extension Ghidra
- [ ] Contribuer à un rapport d'analyse publique

Ressources Recommandées

Ghidra Docs: ghidra-sre.org | The Ghidra Book (No Starch Press) | RE for Beginners (Dennis Yurichev, gratuit)
Platforms CTF: crackmes.one, root-me.org, picoCTF, HackTheBox | MalwareBazaar (samples) | VirusTotal
YouTube: 0xRick, LiveOverflow, MalwareAnalysisForHedgehogs, GynvaelColdwind | Cours: OpenSecurityTraining2

Pour approfondir ce sujet, consultez notre outil open-source malware-analysis-toolkit qui facilite l'analyse automatisée de malwares.

Questions frequentes

Comment mettre en place Ghidra dans un environnement de production ?

La mise en place de Ghidra en production necessite une planification rigoureuse, incluant l'evaluation des prerequis techniques, la definition d'une architecture cible, des tests de validation approfondis et un plan de deploiement progressif avec des points de controle a chaque etape.

Pourquoi Ghidra est-il essentiel pour la securite des systemes d'information ?

Ghidra constitue un element fondamental de la securite des systemes d'information car il permet de reduire significativement la surface d'attaque, d'ameliorer la detection des menaces et de renforcer la posture globale de securite de l'organisation face aux cybermenaces actuelles.

Faut-il des connaissances en assembleur pour pratiquer Ghidra : Guide de Reverse Engineering pour Débutants ?

Des bases en x86/x64 sont nécessaires pour le reverse natif. Pour le .NET ou Java, la décompilation produit du code lisible et l'assembleur est moins critique. Commencez par le langage que vous maîtrisez.

Sources et références : MITRE ATT&CK · CERT-FR

14. Conclusion : premiers pas vers la maîtrise

Ghidra a démocratisé le reverse engineering en mettant entre les mains de tout analyste un outil de calibre professionnel, gratuitement. Les fonctionnalités que nous avons couvertes dans ce guide -- le CodeBrowser, le décompilateur P-Code, les cross-references, le scripting, les plugins -- ne représentent que la surface de ce que Ghidra permet. Avec de la pratique, vous développerez une intuition pour naviguer dans le code binaire, reconnaître les patterns d'obfuscation, et reconstruire la logique des programmes les plus complexes.

Le reverse engineering est une compétence qui se développe par la pratique. Commencez par des crackmes simples, progressez vers des CTF plus complexes, puis attaquez l'analyse de malwares réels dans un environnement isolé. Chaque binaire analysé vous apprend quelque chose de nouveau -- un pattern de code, une technique d'obfuscation, une structure de données. La combinaison de Ghidra avec des outils complémentaires comme Volatility 3 pour l'analyse mémoire, les frameworks C2 pour comprendre l'infrastructure d'attaque, et les frameworks d'IA pour l'analyse de malwares vous donnera une vision complète de la chaîne d'analyse.

Le prochain article de cette série abordera l'analyse des fileless malwares, où Ghidra est utilisé en complément de l'analyse mémoire pour reconstruire les payloads malveillants qui ne touchent jamais le disque.

Ayi NEDJIMI Consultants — Expert cybersécurité offensive & intelligence artificielle

ayinedjimi-consultants.fr · ayi@ayinedjimi-consultants.fr

© 2026 — Reproduction interdite sans autorisation.